

# Hybrid Mobile Applications - ICD0018

TalTech IT College, Andres Käver, 2021-2022, Fall semester

Email: [andres.kaver@taltech.ee](mailto:andres.kaver@taltech.ee)



# React N Class - State

2

- ▶ Local state within component

```
interface State {  
  level: number;  
}  
  
export class Hello extends React.Component<Props, State>{  
  
  constructor(props: Props) {  
    super(props);  
    this.state = { level: props.level }  
  }  
  
  onIncrement = () => this.setState({ level: this.state.level + 1 });  
  onDecrement = () => this.setState({ level: this.state.level - 1 });  
  
  render() {.....
```

# React N Class - State

3

- ▶ Local state within component

```
render() {  
  return (  
    <View style={styles.root}>  
      <Text style={styles.greeting}>  
        Hello, {this.props.name}! Level: {this.state.level}!  
      </Text>  
      <View>  
        <View>  
          <Button title="-" onPress={this.onDecrement} />  
        </View>  
        <View>  
          <Button title="+" onPress={this.onIncrement} />  
        </View>  
      </View>  
    </View>  
  );  
}
```



# React N Functional - State

4

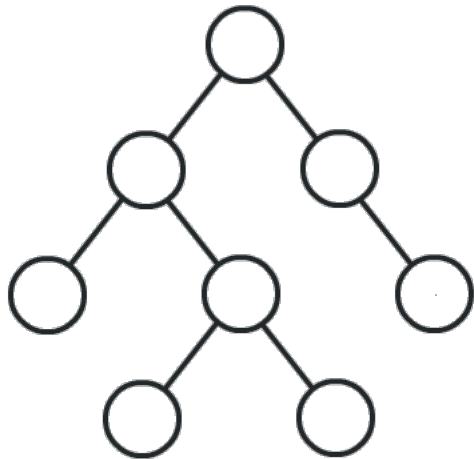
- State within functional component - hook

```
const HelloFn = (props: Props): JSX.Element => {  
  const [counter, setCounter] = useState(0);  
  return (  
    <View style={styles.root}>  
      <Button title="-" onPress={() => setCounter(counter - 1)} />  
      <Text style={styles.greeting}>  
        {props.label} {counter}  
      </Text>  
      <Button title="+" onPress={() => setCounter(counter + 1)} />  
    </View>  
  );  
};
```

# React N – State

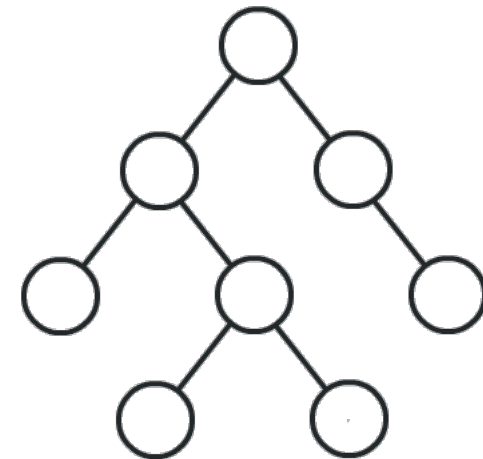
5

- ▶ Callbacks/property drilling

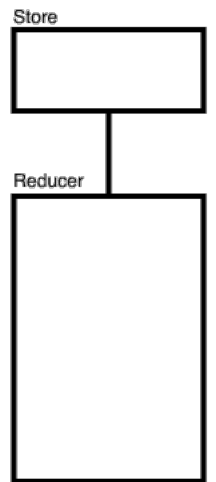


- State change initiated
- State change

- ▶ Context API/Redux



- State change initiated
- State change

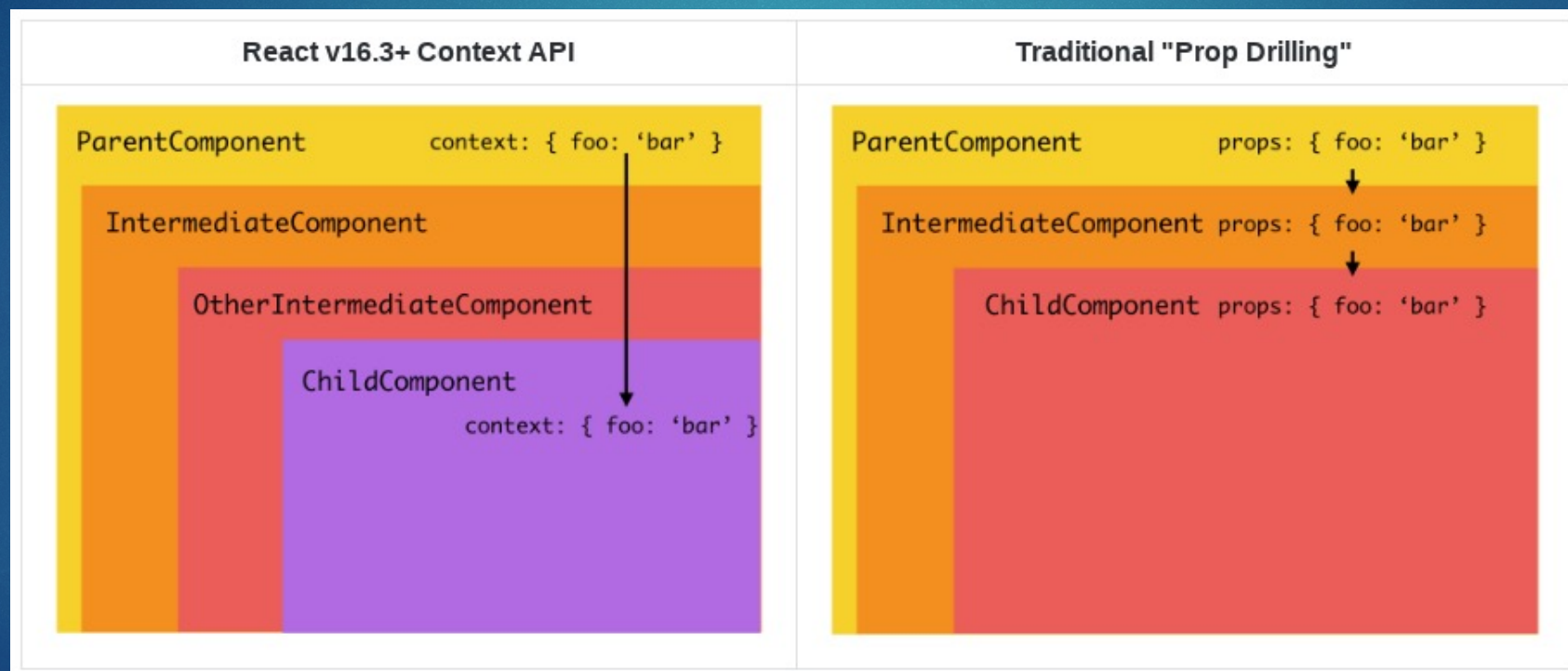




# React N – State

6

## ► Global/Context API vs Prop Drilling





# React N – Context API

7

- ▶ When to Use Context
  - ▶ Context is designed to share data that can be considered “global” for a tree (or subtree) of React components
  - ▶ Context is primarily used when some data needs to be accessible by many components at different nesting levels.



# React N – Context API

8

- ▶ `export const AppContext = React.createContext<IAppContext>(defaultValue);`
  - ▶ Creates context (state) object
- ▶ Also creates Provider and Consumer

```
export const AppContextProvider = AppContext.Provider;  
export const AppContextConsumer = AppContext.Consumer;
```



# React N – Context API

9

## ► Context Provider

- Every Context object comes with a Provider React component that allows consuming components to subscribe to context changes.

```
render() {  
  return (  
    <AppContextProvider value={this.state} >  
      <View style={styles.main}>
```

- defaultValue from createContext is only used, when provider is not found!
- All consumers that are descendants of a Provider will re-render whenever the Provider's **value prop changes**. (same algorithm as Object.is)



# React N Class– Context API

10

```
render() {  
  return (  
    <AppContextConsumer>  
      {appContext =>  
        <View style={styles.mainView}>  
          <Text>Board {appContext.moves}</Text>  
        </View>  
      }  
    </AppContextConsumer>  
  )  
}
```

## ► Context Consumer

- A React component that subscribes to context changes.
- The function receives the current context value and returns a React node. The value argument passed to the function will be equal to the value prop of the closest Provider for this context above in the tree.
- If there is no Provider for this context above, the value argument will be equal to the defaultValue that was passed to createContext().



# React N Functional – Context API

11

- Functional component: hook `useContext(SomeContext)`.

```
import { AppContext } from "../context/appContext";

export interface Props {
  bitNo: number,
}

export const CounterDisplay = (props: Props) => {
  const context = useContext(AppContext);

  return (
    <View style={styles.textContainer}>
      <Text style={styles.text}>{context.getBit(context.counter, props.bitNo)}</Text>
    </View>
  );
}
```



# React N Class – Context API

12

- ▶ `contextType`
- ▶ The `contextType` property on a class can be assigned a Context object created by `React.createContext()`.
- ▶ Lets you consume the nearest current value of that Context type using `this.context`.
- ▶ You can reference this in any of the lifecycle methods including the render function.



# React N Class – Context API

13

- ▶ contextType with TypeScript in class component

```
export class Stats extends React.Component {  
  
  static contextType = AppContext;  
  context!: React.ContextType<typeof AppContext>;  
  
  render() {  
    return (  
      <View style={styles.main}>  
        <Text>Stats: {this.context.moves}</Text>  
      </View>  
    );  
  }  
}
```

# React N Class/Func – Context API

14

- ▶ Mutating context from child components
  - ▶ Include methods for updating the state in the Context object

```
export interface IAppContext {  
  moves: number,  
  changeMoves: (amount: number) => void,  
  setMoves: (amount: number) => void,  
}  
  
export const AppContext = React.createContext<IAppContext>(  
  null  
);
```



# React N Class/Func – Context API

15

```
export interface State extends IAppContext {}
class App extends React.Component<Props, State>{
  constructor(props) {
    super(props);
    this.state = {
      moves: 0,
      changeMoves: this.changeMoves,
      setMoves: this.setMoves,
    }
  }

  changeMoves = (amount: number) => {
    this.setState({ moves: this.state.moves + amount });
  }

  setMoves = (amount: number) => { this.setState({ moves: amount }); }

  render() {
    return (
      <AppContextProvider value={this.state}>
```

# React N – Context API

16

## ▶ Recap

- ▶ Use `ctx = React.createContext()` to create context
- ▶ Pull `ctx.Provider` and `ctx.Consumer` out of `ctx`
- ▶ Wrap `Provider` around parent component
- ▶ Consuming – 3 alternatives
  - ▶ Wrap `<Consumer>{ctx => <></>...}` around context usage.
    - ▶ Use this option, when multiple contexts are needed simultaneously
  - ▶ A class can consume with `static contextType = ctx`
  - ▶ A functional component with `const x = useContext(ctx)`
- ▶ Updating context from child component – include methods in context. Take care of triggering rendering updates (no deep object monitoring)



► THE END!